```
/*
 * CoffeeCan.java
 *
 * This is an OO version of the "Coffee Can" problem first suggested by
 * David Gries in _The Science of Programming_ page 165:
 *
 *  "A coffee can contains some black beans and white beans.  The
 *   following process is repeated as long as possible:
 *
 *   Randomly select two beans from the can.  If they have the same color,
 *   throw them out, but put another black bean in.  (Enough extra black beans
 *   are available to do this.)  If they are of different colors, place the
 *   white one back in the can and throw the black one away.
 *
 *   Execution of this process reduces the number of beans in the can by one.
 *   Repetition of this process must terminate with exactly one bean in the
 *   can, for then two beans cannot be selected.  The question is: what, if
 *   anything, can be said about the color of the final bean based on the
 *   number of white beans and the number of black beans initially in the can?"
 *
 * The class has an invariant which is partially given below.  Discovering the
 * rest of the invariant is the key to answering the question.
 *
 * Each method is documented with preconditions and postconditions.  To avoid
 * saying the same thing twice, information in the @param tags should be
 * considered part of the precondition, and information in the @return tags
 * should be considered part of the postcondition.
 *
 * Copyright (c) 2000 - Russell C. Bjork
 */

import java.io.*;

/** Representation for Gries' coffee can.
 *
 *  Invariant: there is at least one bean in the can and
 *             the number of beans of each color is >= 0 and
 *             ????
 */
public class CoffeeCan
{
    /** Main program.  Repeatedly ask user for maximum number of beans of each
     *  color in can, and then play one instance of the "coffee can" game,
     *  reporting activity to System.out.  Stop when max = 0.
     *
     *  NOTE: By entering a negative value, the user can cause the precondition
     *  of the constructor to be violated.  What happens in this case?
     */
    public static void main(String [] args) throws IOException
    {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Maximum number of beans of each color: ");
        int max = Integer.parseInt(input.readLine());

        while (max != 0)
        {
            // Play one game

            CoffeeCan theCan = new CoffeeCan(max);
            String initialContents = theCan.reportContents();
            System.out.println("Initial contents: " + initialContents);
```

```
        while (theCan.numberOfBeans() > 1)
        {
            System.out.println();
            try
            {
                Thread.sleep (5 * 1000);
            }
            catch (InterruptedException e)
            { }

            String roundResults = theCan.playOneRound();
            System.out.println("Results of round: " + roundResults);

            String currentContents = theCan.reportContents();
            System.out.println("Current contents: " + currentContents);
        }

        String finalColor = theCan.lastBeanColor();
        System.out.println("Color of the final bean is " + finalColor);
        System.out.println();

        // Ask user for parameters for next game, or 0 to quit

        System.out.print("Maximum number of beans of each color - 0 to quit: ");
        max = Integer.parseInt(input.readLine());
    }

    System.exit(0);
}

/** Constructor
 *
 *  Precondition: max >= 1
 *
 *  @param max maximum number of beans of each color that can
 *         initially be in the can
 *
 *  Postconditions: can contains between 1 and max white beans and
 *                  between 1 and max black beans.
 *
 */
public CoffeeCan(int max)
{
    whiteBeans = (int) (1 + max * Math.random());
    blackBeans = (int) (1 + max * Math.random());
}

/** Report current contents of the can
 *
 *  Preconditions: none
 *  Postcondition: can contents are unchanged
 *
 *  @return string describing the current contents of the can
 */
public String reportContents()
{
    return "Can contains: " + whiteBeans +
        " white beans and " + blackBeans + " black beans - total = " +
        (whiteBeans + blackBeans);
}
```

```
/** Report number of beans currently in the can
 *
 *  Preconditions: none
 *  Postconditions: can contents are unchanged
 *
 *  @return total number of beans in can
 */
public int numberOfBeans()
{
    return whiteBeans + blackBeans;
}

/** Play one round of the "coffee can" game.  Draw two beans and put
 *  one back, as described by the rules above.
 *
 *  Precondition: there is more than one bean in the can
 *  Postcondition: the number of beans in the can is reduced by 1, in
 *                 accordance with the rules of the game
 *
 *  @return string describing what took place
 */
public String playOneRound()
{
    String first = chooseBean();
    if (first.equals("White"))
        whiteBeans --;
    else
        blackBeans --;

    String second = chooseBean();
    if (second.equals("White"))
        whiteBeans --;
    else
        blackBeans --;

    String putBack;
    if (first.equals(second))
    {
        putBack = "Black";
        blackBeans ++;
    }
    else
    {
        putBack = "White";
        whiteBeans ++;
    }

    return "Drew: " + first + ", " + second + ".  Put back: " + putBack;
}
```

```
    /** Choose a single bean to draw
     *
     *  Preconditions:  there is at least one bean in the can
     *
     *  Postconditions: Return value is either "White" or "Black",
     *                  and there is at least one bean in the can of that color
     *
     *  @return color of bean to draw
     *
     */
    private String chooseBean()
    {
        if (whiteBeans > 0 && blackBeans > 0)
            if (Math.random() < 0.5)
                return "White";
            else
                return "Black";

        else if (whiteBeans > 0)
            return "White";

        else // must be the case that blackBeans > 0
            return "Black";
    }

    /** Report the color of the final bean
     *
     *  Preconditions: the can contains exactly one bean
     *
     *  Postcondition: can contents are unchanged
     *
     *  @return color of the one bean in the can
     *
     */
    public String lastBeanColor()
    {
        if (whiteBeans == 1)
            return "White";

        else // must be that blackBeans == 1
            return "Black";
    }

    // Number of beans of each kind currently in the can

    private int whiteBeans;
    private int blackBeans;
}
```